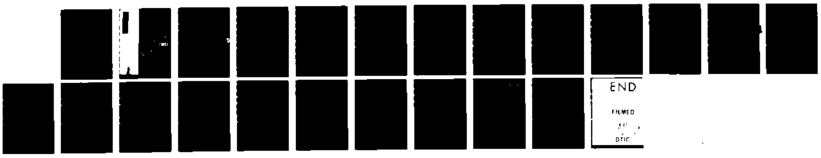
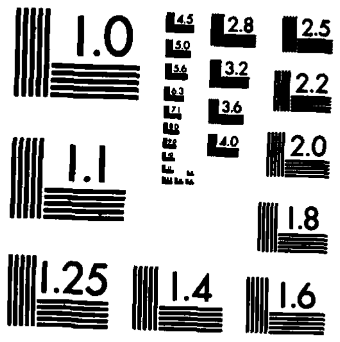
AD-A126 238 EFFICIENT TREE HANDLING PROCEDURES FOR ALLOCATION/PROCESSING NETWORKS(U) TEXAS UNIV AT AUSTIN CENTER FOR CYBERNETIC STUDIES M ENGQUIST ET AL. DEC 82 1/1

UNCLASSIFIED CCS-437 N00014-81-C-0236 F/G 12/2 NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
7.1

(8)

Research Report CCS 437

EFFICIENT TREE HANDLING PROCEDURES FOR ALLOCATION/PROCESSING NETWORKS

by

Michael Engquist
Chou-Hong Chen

CENTER FOR CYBERNETIC STUDIES
The University of Texas
Austin, Texas 78712

(8)

Research Report CCS 437

EFFICIENT TREE HANDLING PROCEDURES FOR ALLOCATION/PROCESSING NETWORKS

by

Michael Engquist
Chou-Hong Chen

December 1982

This paper is supported partly by ONR Contracts N00014-81-C-0236 and N00014-82-K-0295 with the Center for Cybernetic Studies, The University of Texas at Austin and partly by a grant from the CBA Foundation, The University of Texas at Austin. Reproduction in whole or in part is permitted for any purpose of the United States Government.

CENTER FOR CYBERNETIC STUDIES

A. Charnes, Director
Business Economics Building, 203E
The University of Texas at Austin
Austin, Texas 78712
(512) 471-1821

- A -

ABSTRACT

Processing network problems are minimum cost network flow problems in which the flow entering (or leaving) a node may be constrained to do so in given proportions. These problems include many practical large-scale linear programming applications. In this paper a special class of processing network problems called allocation/processing (AP) network problems is described. AP network problems model a realistic situation in which a single raw material is allocated to factories and the resulting finished products are distributed to customers.

A special partitioning method (PPR algorithm) is introduced for the solution of AP network problems. This algorithm maintains a working basis as well as a so-called representative spanning tree. A method for updating the representative spanning tree is discussed which avoids tracing a number of cycles at each iteration.

A FORTRAN implementation (PPRNET) of the PPR algorithm is described. Comparative computational results are presented for randomly generated AP network problems which were solved by PPRNET and MINOS. These results indicate that significant computational gains are possible with the use of special purpose processing network codes.

Key Words: Network Flow Algorithms, Linear Programming, Large Scale Systems, Processing Networks.

1. INTRODUCTION

Processing network problems are a special type of embedded network problem. For these problems, the network has two special types of nodes called refining nodes and blending nodes. At a refining node, the entering flow splits into several leaving components in fixed proportions while at a blending node, several components enter in fixed proportions to form the leaving flow. The term processing node is used to describe either a refining node or a blending node. A number of applications which exhibit processing network structure have been listed by Koene [18], including energy modelling [20] and assembly models [25]. McBride [21] notes that short-term financial planning models [19] can be formulated as processing network problems. Charnes et al. [5] have recently described a processing network model which is used in manpower planning.

The computational study [11] showed that special purpose network codes are 150-200 times faster than the APEX III linear programming (LP) code. This raises the question of whether it is possible to develop special purpose processing network codes which are significantly faster than state-of-the-art LP codes. We believe the answer to this question is in the affirmative, and we present computational results in this paper which support this position.

Although processing network problems can be solved by embedded network techniques as described by Glover and Klingman [12] and Gupta and McBride [13], there have been specialized algorithms proposed by Koene [18] and McBride [21] which take advantage of processing network structure. McBride's algorithm has been implemented, and some preliminary computational results are presented in [21]. Charnes et al. [5] describe a heuristic procedure for the solution of processing networks. Beck, Lasdon and Engquist [4] solve network problems with equal flow constraints by means of a penalty method. In this paper we describe yet another algorithm for solving certain processing network problems, and we discuss its implementation and testing.

2. TEST PROBLEMS

All processing nodes for the class of problems for which computational tests were conducted are refining nodes. For processing node j, as shown in Figure 1, the flow on an arc (j,k) must be a fraction α_{jk} of the flow entering node j. Since conservation of flow must hold at node j, it is required that

$$\sum_{v=1}^{p} \alpha_{jk(v)} = 1 \tag{2.1}$$

We note that processing nodes are represented graphically as squares while ordinary nodes are shown as circles. In Figure 1, we refer to the arcs leaving node j as processing arcs and to the arc entering node j as an allocation arc.

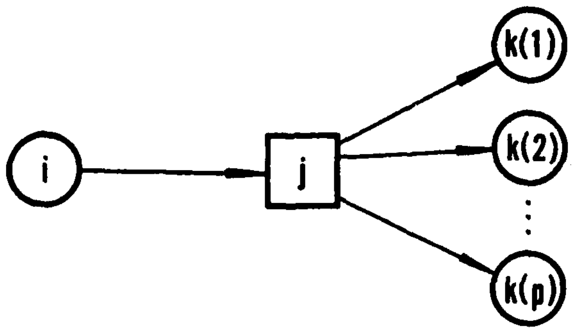


Figure 1. A Processing Node.

Processing network problems can be formulated as networks with either side constraints or side variables, and we proceed to describe these two formulations as they apply to our class of test problems. The first of these, called the <u>sparse formulation</u>, is used by our solution algorithm which is described in Section 4. In addition to the usual conservation of flow constraint at processing node j of Figure 1, the sparse formulation requires

$$\beta_{jk(v)}\, x_{jk(v)} - \beta_{jk(v+1)}\, x_{jk(v+1)} = 0, \quad v=1,2,\ldots,p-1 \quad (2.2)$$

where $\beta_{jk(v)} = \alpha^{-1}_{jk(v)}$ and $x_{jk(v)}$ is the flow on arc (j,k(v)).

The second formulation is known as the <u>compact formulation</u>. In this formulation, the processing arcs of Figure 1 are replaced by a single processing column in the constraint matrix of the form

$$\begin{array}{ll} 1 & \text{in row } j \\ -\alpha_{jk(v)} & \text{in row } k(v),\ v=1,2,\ldots,p \\ 0 & \text{elsewhere.} \end{array} \quad (2.3)$$

In the computational tests described in Section 5, the compact formulation is used to generate an input file for the MINOS optimization code of Murtagh and Saunders [22].

A further requirement for our test problems is that all allocation arcs emanate from a common node. These test problems are called <u>allocation/processing</u> (AP) <u>network problems</u>, and they may be described as follows: A certain amount (s) of raw material is supplied at the allocation node. This raw material is to be allocated to factories (processing nodes) which all produce the same finished products from the raw material. However, the proportion of a unit of raw material devoted to a particular product varies from factory to factory. After a finished product leaves the factory, it enters a transportation subnetwork devoted to distribution of this type of product. All such transportation subnetworks are identical and their destination nodes have customer demands (d_i) which must be satisfied.

The topology of an AP network is determined by the following parameters: the number of processing nodes r, the number of finished products p, and the number of destinations q in a transportation subnetwork. An example of an AP network with r = 2, p = 2, and q = 2 is shown in Figure 2. In this example, node 1 is the allocation node, nodes 2 and 3 are the processing nodes, and nodes 4 through 11 are contained in the transportation subnetworks.

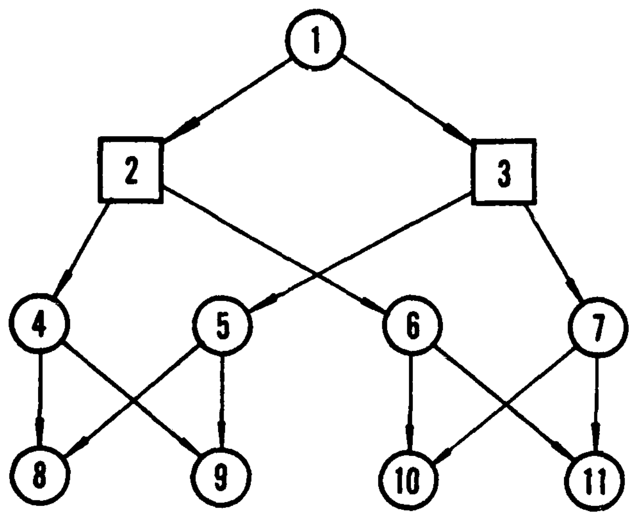


Figure 2. An AP Network Example.

The allocation and processing arcs in AP problems are capacitated while all other arcs are uncapacitated. The capacities on the allocation arcs model the limited throughput of the factories. No capacities were imposed on the arcs of the transportation subnetworks since the density of these subnetworks indicates abundant means of transportation.

It follows from the theory of transportation problems that for any feasible solution of an AP network problem, the total flow into a transportation subnetwork must be equal to the total demand for that subnetwork. The amount of flow into a given transportation subnetwork is also equal to a sum of terms of the form $\alpha_{jk} \times$ (flow on corresponding allocation arc). This gives rise to p equations in r unknowns. In problems where the α_{jk} are randomly generated and $r < p$, this system of equations will have a unique solution almost surely (if the d_i are chosen to guarantee that the system is consistent). If such a unique solution exists, the AP network problem can be decomposed into p transportation problems. For this reason, we restricted computational tests to AP networks with $p < r$.

3. BASIS STRUCTURE

We now consider the LP problem which corresponds to an AP network problem with parameters p, q, and r using the sparse formulation.

$$\begin{aligned} &\text{minimize} \quad cx \\ &\text{subject to} \quad Ax = b \qquad (3.1) \\ &\qquad 0 \le x \le u \end{aligned}$$

In (3.1), A is an $m \times n$ matrix; c, x, and u are n-vectors; and b is an m-vector. If an arc is to be uncapacitated this is accomplished by choosing a sufficiently large component of u for that arc. Without loss of generality, it is assumed for each processing node as shown in Figure 1,

$$\alpha_{jk(v)} u_{ij} = u_{jk(v)} \text{ , } v = 1,2,\ldots,p. \qquad (3.2)$$

The matrix A in (3.1) can be written

$$A = \begin{bmatrix} A_1 & A_2 \\ 0 & A_3 \end{bmatrix} \qquad (3.3)$$

If we let $m_1 = m - r(p - 1)$ and $n_1 = n - rp$, then the $m_1 \times n_1$ matrix A_1 is the node-arc incidence matrix for the pure network (non-processing) arcs of the problem. The $m_1 \times rp$ matrix A_2 is the node-arc incidence matrix for the processing arcs, and the $r(p - 1) \times rp$ matrix A_3 contains the additional proportional flow constraints of (2.2). We assume without loss of generality that an extra column containing a single 1 with all other components equal 0 (root arc column) has been adjoined to A_1 and that the rank of A_1 is m_1. It is easily deduced that the rank of A_3 is $r(p - 1)$ and hence that the rank of A is m.

From now on, B denotes a basis of (3.1). Thus, B is a nonsingular $m \times m$ submatrix of A.

Each processing node j as shown in Figure 1 has p processing arcs emanating from it. It follows from (2.2) that B contains either p or p-1 of the columns corresponding to these processing arcs. If, for processing node j, B contains p processing arcs, then processing node j is said to be active. Otherwise, processing node j is inactive. If a spanning tree of the AP network has the property that it contains a single processing arc for each active processing node, it is called a representative spanning tree (RS-tree). If, further, all allocation arcs are included in an RS-tree, we call such a tree a representative spanning allocation tree (RSA-tree). In either case, the processing arc corresponding to processing node j is said to be the representative arc of node j.

Theorem 1 [18]. For every basic feasible solution of (3.1), there exists a corresponding basis B such that

$$B = \begin{bmatrix} T & C \\ D & F \end{bmatrix} \tag{3.4}$$

where the $m_1 \times m_1$ matrix T is the node-arc incidence matrix of an RS-tree and the $m_1 \times r(p-1)$ matrix C contains columns of A_2.

Corollary. In Theorem 1, we may conclude that T is the node-arc incidence matrix of an RSA-tree.

Proof. Starting with B from Theorem 1, we can make degenerate pivots using (3.2) until a new B is obtained for which T in (3.4) is as desired.

The following result is useful whenever a partitioned basis matrix is maintained. A proof may be found in [17].

Theorem 2. If the matrices of (3.4) are given and the matrix Q is defined by

$$Q = F - DT^{-1}C \, , \tag{3.5}$$

then Q is nonsingular and

$$B^{-1} = \begin{bmatrix} T^{-1} + T^{-1}CQ^{-1}DT^{-1} & -T^{-1}CQ^{-1} \\ -Q^{-1}DT^{-1} & Q^{-1} \end{bmatrix} \tag{3.6}$$

The tree structure associated with T allows computations involving T^{-1} to be carried out using highly efficient tree labeling procedures [2] so that T^{-1} need not be explicitly maintained. The matrix Q^{-1} must also be maintained in some form in order to carry out the computations required by the simplex method. However, Q is an $r(p - 1)$ dimensional matrix, and for purposes of computational testing, we maintained a smaller matrix R which has dimension r. The matrix R is constructed from Q by using certain row operations which utilize the special structure that Q inherits from (2.2). We omit the details of this construction but we note that in our computational testing the matrix R was kept in dense form along with its inverse. The multiplication of a row or column vector by Q^{-1} is replaced by a corresponding multiplication involving R^{-1}. In other words, for the algorithm described in the next section, R plays the role of a working basis.

4. A SIMPLEX VARIANT AND ITS IMPLEMENTATION

The steps of the revised simplex algorithm are well known. These steps can be specialized to a situation in which a partitioned basis is maintained for a network problem with side constraints. Such a specialization is described, for example, in Chapter 7 of [17]. In this section, we point out some of the differences between our algorithmic approach and that of [17]. Our algorithm, which is restricted to AP network problems, may be described as a partitioning algorithm for certain processing network problems (PPR algorithm). We also discuss a FORTRAN implementation of this algorithm called PPRNET.

It will be useful to make the convention for AP networks that the allocation node is node 1 while the processing nodes are nodes 2 through $r + 1$.

In the initialization step of the PPR algorithm, we set up an initial RSA-tree and the corresponding matrix R. This RSA-tree is constructed using the minimum average cost procedure which is described after we introduce some notation. The forward set $F(u)$ of a node u in a network is defined to be the set of all nodes v such that an arc (u,v) of the network exists. The average cost $\hat{c}_j$ of processing node j in an AP network is defined by

$$\hat{c}_j = c_{1j} + \frac{1}{q} \sum_{k \in F(j)} \alpha_{jk} \sum_{u \in F(k)} c_{ku} . \qquad (4.1)$$

It should be noted that (4.1) represents the expected cost incurred by a unit flow through node j when the fraction of this flow reaching a transportation subnetwork is equally likely to take any available arc. The procedure for obtaining the minimum average cost RSA-tree follows.

(i) Select processing node j′ for which

$$\hat{c}_{j'} = \min_{2 \leq j \leq r+1} \hat{c}_j. \tag{4.2}$$

(ii) Introduce p artificial arcs into the RSA-tree each emanating from the allocation node and terminating at a distinct node of F(j′). The flow on each such arc equals the total demand for the transportation subnetwork with which it is incident. These arcs are uncapacitated with a Big-M cost.

(iii) For each $k \in F(j')$, include q arcs (k,u), $u \in F(k)$ in the RSA-tree. Arc (k,u) has flow d_u.

(iv) For each origin node k of a transportation subproblem which is not in F(j′), choose an arc (k,u′) such that

$$c_{ku'} = \min_{u \in F(k)} c_{ku} . \tag{4.3}$$

These arcs are included in the RSA-tree, with zero flow.

(v) Include all allocation arcs in the RSA-tree.

To accompany this RSA-tree, it is necessary to specify for each processing node a processing arc which will be nonbasic (all processing nodes are initially inactive). These processing arcs can be chosen arbitrarily. Once they are chosen, the matrix R, which is initially diagonal, is determined.

Pricing by the PPR algorithm is done by first pricing the processing arcs and then the pure network arcs. The nonbasic processing arcs are included in a candidate list which is maintained with a threshold strategy [9]. According to this strategy, on the first pass of the candidate list, the pivot eligible arc (j′,k′) with the largest absolute reduced cost is taken as the entering arc. On subsequent passes, an entering arc (j,k) must satisfy

$$|\bar{c}_{jk}| > \gamma|\bar{c}_{j'k'}| \tag{4.4}$$

where $\bar{c}_{jk}$ and $\bar{c}_{j'k'}$ denote reduced costs and the parameter γ is a fraction. In PPRNET, 0.5 is the value of γ. When the candidate list yields no more pivot eligible arcs, the pure network arcs are priced using the outward node most negative rule [10], [24]. After the pure network arcs have been priced, the processing arcs are again priced, and so on until no pivot eligible arcs remain.

We note that when the ratio test is carried out, the allocation arcs are not considered since they never leave the basis. This feature of the PPR algorithm is made possible by (3.2).

The basis updating procedure for the PPR algorithm is quite different from that of the algorithm described in [17]. The RSA-tree structure must be maintained, and there are two cases to be considered. In the first case, the leaving arc lies on the cycle formed in the RSA-tree by the entering arc, and no allocation arcs lie on this cycle. In this case, the RSA-tree is updated as in an ordinary network pivot and the matrix R needs no updating. In the remaining case, it may be required that several subtrees be rehung. We consider the current set of representative arcs to be a matching between active processing nodes and subtrees. The set of subtrees is changed when the leaving arc is deleted and the entering arc is adjoined. If we put a cost of 0 on the current representative arcs and a cost of 1 on all other processing arcs, then the problem of updating the RSA-tree with a minimum number of changes in representative arcs becomes one of finding a minimum cost augmenting path. Augmenting path methods are covered in [8].

In order to obtain an augmenting path, PPRNET uses Dial's implementation [6] of the shortest path algorithm of Dijkstra [7]. Once the RSA-tree has been updated, the matrix R is regenerated. PPRNET then computes R^{-1} using the IMSL [16] subroutine LINV1F. The matrix R^{-1} is handled in this way since PPRNET is intended for experimental use only.

The data structure of PPRNET is patterned after those of pure network codes. The RSA-tree has the predecessor, depth, thread and reverse thread [2] associated with it. Additional arrays are used to carry out the operations required in using the partitioned basis inverse (3.6). PPRNET uses five arrays of length n; however, two of these were included for convenience and could be eliminated without affecting the performance of the code. Also included are eleven arrays of length m_1, eight arrays of length pr, eleven arrays of length r, and the two $r \times r$ matrices R and R^{-1}. The use of network data structures in processing network codes is important since it allows extremely large problems to be solved in main memory.

5. COMPUTATIONAL TESTING

A FORTRAN code called PRGEN was developed which generates random AP network problems. PRGEN creates two files--corresponding to the two problem formulations--for a given problem. In order to insure that the problems generated are feasible, PRGEN creates a feasible flow as it builds the network. The capacity on an allocation arc is set to be μ times the feasible flow generated for that arc. The value μ, which is at least one, represents the excess capacity of the factory system, and it is an input parameter for PRGEN. For a given processing node j, all α_{jk} but one are chosen randomly from a certain problem dependent range with the remaining value chosen so that condition (2.1) holds.

MINOS [22], is designed to solve linearly constrained problems with either linear or nonlinear objectives. However, we only made use of the linear programming capabilities of MINOS. MINOS uses the revised simplex method with Bartels-Golub updating [3] of a factored basis inverse. The basis reinversion routine is derived from the method of Hellerman and Rarick [14], [15]. All parameters for MINOS were set to default values [23] except for the partial price parameter. This parameter, which we denote as k in Table 2, is used to partition the nonbasic variables into k subsets of equal size for purposes of pricing. The starting basis for MINOS is created by a routine which selects from all columns of the constraint matrix.

For PPRNET, in all occurrences where testing for zero was required, a tolerance of 10^{-4} was used. The Big-M value used in PPRNET was 99999. Although no numerical difficulties were encountered with the Big-M Method, it seems wise to replace it in future testing with a Phase I, Phase II approach or with an exact non-Archimedean approach such as in the NONARC code [1].

The parameter values for the seven problem groups tested are shown in Table 1. The number of problems in each group is also shown. Groups four and five contain identical problems except that the capacities on the allocation arcs are larger in group five. The costs for the problems generated are randomly chosen and for the allocation arcs they are between 500 and 1000 while for the arcs in the transportation subnetworks they lie between 1 and 100. The costs on processing arcs are zero. The total supply for these problems varied from 500 to 900.

The test problems were solved using the MNF compiler on the CDC Dual Cyber 170/750 at the University of Texas. The results as shown in Table 2 are averages for each problem group. The execution times reported are in central processor seconds and are exclusive of input and output. Invert time is the total time required by the LINV1F subroutine in computing R^{-1}.

The non-network iterations are those iterations of the PPR algorithm which involve the regeneration and reinversion of R. Also reported in Table 2 is the number of processing nodes active at optimality.

It should be noted that subsequent to the generation of the test results in Table 2, we found that MINOS times could be decreased by about 25% in problem groups one through five by setting the partial price parameter to 30. Presumably, a similar decrease is possible in group six as

TABLE 1

AP NETWORK PROBLEM DATA

Problem Group	No. Problems	r	p	q	μ	Compact Formulation No. Rows	Compact Formulation No. Columns
1	5	10	5	40	1.1	261	2020
2	5	16	8	16	1.1	273	2080
3	5	20	10	10	1.1	321	2040
4	5	20	5	20	1.1	221	2040
5	5	20	5	20	1.5	221	2040
6	2	20	15	15	1.1	546	4540
7	1	20	8	50	1.1	581	8040

TABLE 2

PERFORMANCE STATISTICS FOR AP NETWORK PROBLEMS

Problem Group	Method	Total Time*	Invert Time	Total Iterations	Non-network Iterations	Active Proc.Nodes
1	MINOS, k=2	49.2		1040		
1	PPRNET	14.5	3.6	1548	910	5
2	MINOS, k=2	70.6		1325		
2	PPRNET	36.7	15.4	2832	1385	8.2
3	MINOS, k=2	77.6		1328		
3	PPRNET	38.5	18.4	2381	997	10
4	MINOS, k=3	52.2		1155		
4	PPRNET	31.3	18.1	2022	985	5.2
5	MINOS, k=3	53.6		1141		
5	PPRNET	40.2	23.5	2506	1269	11
6	MINOS, k=7	256.0		2665		
6	PPRNET	138.3	52.0	6653	2782	15
7	MINOS	DID	NOT	RUN		
7	PPRNET	276.7	113.6	12208	6128	8

* See text for reduced MINOS times

well. When this decrease is accounted for, the ratio of total MINOS times to total PPRNET times ranges from about 2.5 for problem group one down to 1.0 for problem group five. Although several parameters are varied over this range of problems, it seems clear that the number of processing nodes r plays a major role in determining PPRNET solution times. In examining these solution times, one should keep in mind that for each non-network iteration it is necessary for PPRNET to regenerate and reinvert the $r \times r$ matrix R. For more advanced processing network codes, the time spent in maintaining the working basis inverse should be significantly decreased. The number of processing nodes active at optimality varies from 40% to 75% of the existing processing nodes. This is of interest since for the algorithms described in [12],[18],[21] the size of the working basis depends on the number of active processing nodes.

6. CONCLUDING REMARKS

We have presented encouraging computational results for a specialized optimization code for solving certain processing network problems. These results indicate that it is possible to achieve significant computational gains for problems in which the non-network portion is relatively small. The need for developing such codes is manifested by the widespread applications which are possible.

REFERENCES

1. A. Ali, Private Communication, 1982.

2. R. Barr, F. Glover and D. Klingman, "Enhancements of Spanning Tree Labeling Procedures for Network Optimization," INFOR 17 (1979) 16-34.

3. R. Bartels and G. Golub, "The Simplex Method of Linear Programming Using LU Decomposition," Communications of the ACM 12 (1969) 266-268.

4. P. Beck, L. Lasdon and M. Engquist, "A Reduced Gradient Algorithm for Nonlinear Network Problems," to appear in ACM Transactions on Mathematical Software.

5. A. Charnes, W. W. Cooper, D. Divine, W. Hinkel, J. Koning and V. Lovegren, "A Sea-shore Rotation Goal Programming Model for Navy Use," Research Report CCS 429, Center for Cybernetic Studies, The University of Texas, Austin (1982).

6. R. Dial, "Algorithm 360 Shortest Path Forest with Topological Ordering," Communications of the ACM 12 (1969) 632-633.

7. E. Dijkstra, "A Note on Two Problems in Connexion with Graphs," Numerical Mathematics 1 (1959) 269-271.

8. J. Edmonds and R. Karp, "Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems," Journal of the ACM 19 (1972) 248-264.

9. D. Gibby, Private Communication, 1978.

10. F. Glover, D. Karney, D. Klingman and A. Napier, "A Computational Study on Start Procedures, Basis Change Criteria, and Solution Algorithms for Transportation Problems," Management Science 20 (1974) 793-813.

11. F. Glover and D. Klingman, "Capsule View of Future Developments on Large Scale Network and Network-Related Problems," Research Report CCS 238, Center for Cybernetic Studies, The University of Texas, Austin (1975).

12. F. Glover and D. Klingman, "The Simplex/SON Algorithm for LP/Embedded Network Problems," Mathematical Programming Study 15 (1981) 148-176.

13. A. Gupta and R. McBride, "Solving Embedded Generalized Network Problems," Presented at TIMS/ORSA Conference, Detroit (1982).

14. E. Hellerman and D. Rarick, "Reinversion with the Preassigned Pivot Procedure," Mathematical Programming 1 (1971) 195-216.

15. E. Hellerman and D. Rarick, "The Partitioned Preassigned Pivot Procedure," in: D. J. Rose and R. A. Willoughby, eds., Sparse Matrices and Their Applications, Plenum Press, New York (1972) 67-76.

16. IMSL Reference Manual, Ninth Edition, IMSL,Inc., Houston (1982).

17. J. Kennington and R. Helgason, Algorithms for Network Programming, John Wiley and Sons, New York (1980).

18. J. Koene, "Minimal Cost Flow in Processing Networks, a Primal Approach," Ph.D. Thesis, Eindhoven University of Technology, Eindhoven (1982).

19. S. Maier and J. Weide, "A Practical Approach to Short-run Financial Planning," in: K. Smith, ed., Readings on the Management of Working Capital, West Publishing Co., St. Paul (1980) 525-533.

20. A. Manne, R. Richels and J. Weynant, "Energy Policy Modelling: A Survey," Operations Research 27 (1979) 1-36.

21. R. McBride, "Solving Network Problems with Proportional Constraints," Presented at ORSA/TIMS Conference, San Diego (1982).

22. B. Murtagh and M. Saunders, "Large Scale Linearly Constrained Optimization," Mathematical Programming 14 (1978) 41-72.

23. B. Murtagh and M. Saunders, "MINOS User's Guide," Technical Report SOL 77-9, Systems Optimization Laboratory, Department of Operations Research, Stanford University, Stanford (1977).

24. V. Srinivason and G. Thompson, "Benefit-Cost Analysis of Coding Techniques for the Primal Transportation Algorithm," Journal of the ACM 20 (1973) 194-213.

25. E. Steinberg and H. Napier, "Optimal Multi-level Lot Sizing for Requirements Planning Systems," Management Science 26 (1980) 1258-1271.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER CCS 437	2. GOVT ACCESSION NO. AD-A126238	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) EFFICIENT TREE HANDLING PROCEDURES FOR ALLOCATION/PROCESSING NETWORKS		5. TYPE OF REPORT & PERIOD COVERED 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) M. Engquist and C.-H. Chen		8. CONTRACT OR GRANT NUMBER(s) N00014-81-C-0236 N00014-82-K-0295
9. PERFORMING ORGANIZATION NAME AND ADDRESS Center for Cybernetic Studies The University of Texas at Austin Austin, Texas 78712		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research (Code 434) Washington, D.C.		12. REPORT DATE December 1982 13. NUMBER OF PAGES 21
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

This document has been approved for public release and sale; its distribution is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Network Flow Algorithms, Linear Programming, Large Scale Systems, Processing Networks

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Processing network problems are minimum cost network flow problems in which the flowing entering (or leaving) a node may be constrained to do so in given proportions. These problems include many practical large-scale linear programming applications. In this paper a special class of processing network problems called allocation/processing (AP) network problems is described. AP network problems model a realistic situation in which a single raw material is allocated to factories and the resulting finished products are distributed to customers.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601 |

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. ABSTRACT (continued)

A special partitioning method (PPR algorithm) is introduced for the solution of AP network problems. This algorithm maintains a working basis as well as a so-called representative spanning tree. A method for updating the representative spanning tree is discussed which avoids tracing a number of cycles at each iteration.

A FORTRAN implementation (PPRNET) of the PPR algorithm is described. Comparative computational results are presented for randomly generated AP network problems which were solved by PPRNET and MINOS. These results indicate that significant computational gains are possible with the use of special purpose processing network codes.

4 - 8

DTI